Leveraging Large Language Models to Enhance Machine-Learning-Driven HPC Job Scheduling

Kshitij Bhardwaj
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
Livermore, CA 94550
bhardwaj2@llnl.gov

Torrey Wagner
Dept. of Systems Engineering & Management
Air Force Institute of Technology
WPAFB, OH 45433
torrey.wagner.2@us.af.mil

Edgar A. León
Livermore Computing
Lawrence Livermore National Laboratory
Livermore, CA 94550
leon@llnl.gov

Abstract

High-performance computing (HPC) systems rely on job schedulers like Slurm to allocate compute resources to submitted workloads. Recently, machine learning models have been used to predict job runtimes which can be used by schedulers to optimize utilization. However, many of these models struggle to effectively encode string-type job features, typically relying on integer-based Label or One-hot encoding methods. In this paper, we use Transformer-based large language models, particularly Sentence-BERT (SBERT), to semantically encode job features for regression-based job runtime prediction. Using a 90,000-record 169-feature Slurm dataset we evaluate four SBERT variants and compare them against traditional encodings using four regression models. Our results show that SBERT-based encodings—especially using the all-MiniLM-L6-v2 model—substantially outperform conventional methods, achieving an r^2 score up to 0.88; $2.3\times$ higher than traditionally-used Label encoding. Moreover, we highlight practical trade-offs, such as model memory size versus accuracy, to guide the selection of efficient encoders for production HPC systems.

1 Introduction

Supercomputers are crucial for large parallel workloads, with schedulers such as Slurm assigning nodes by job size and priority. As demand grows, insufficient idle nodes force jobs to wait. When compute demand exceeds supply, a policy must choose which job runs next. First-Come, First-Serve (FCFS) is common, but if the head-of-queue job cannot start, resources sit idle instead of serving smaller, later jobs. Backfilling addresses this by running jobs that fit current capacity, but depends on accurate runtime estimates which are difficult to provide Srinivasan et al. [2002].

Automated prediction generally follows two paths: (i) application/binary analysis—often costly or infeasible—and (ii) learning from historical logs. Prior work shows that Random Forest, XGBoost, and Neural Networks can predict runtimes from features such as user/partition, CPUs, tasks, memory, QoS, and time limits Menear et al. [2023], Vercellino et al. [2023], Menear et al. [2024], Yang et al. [2023]. Many of these inputs are strings (e.g., Command, PartitionName, Licenses) and must be numerically encoded; standard integer-based Label/Categorical Menear et al. [2024] or One-hot encodings Lamar et al. [2023] are fast but ignore semantic relationships between different features.

We instead encode job metadata with Transformer-based encodings (such as using Sentence-Bert or SBERT Reimers and Gurevych [2019]) and feed them to standard regressors. SBERT yields semantically meaningful vectors so similar names, users, or groups map nearby, revealing batch/workload patterns that integer encodings miss. We evaluate four SBERT variants—`all-MiniLM-L6-v2` Reimers [2021a], `all-MiniLM-L12-v1` Reimers [2020a], `all-mpnet-base-v2` Reimers [2020b], and `xlm-r-base-en-ko-nli-ststb` Reimers [2021b]—against Label and One-hot encodings. Our

ML for Systems Workshop, Conference on Neural Information Processing Systems (NeurIPS 2025).

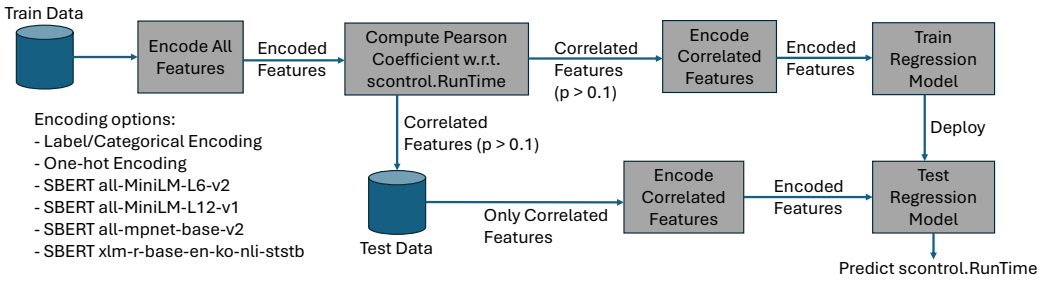

Figure 1: Overview of our runtime prediction pipeline.

results show that all-MiniLM-L6-v2 model substantially outperforms conventional methods, achieving an r^2 2.3× higher than Label encoding.

2 Proposed Framework

Figure 1 shows the details of our proposed job runtime prediction framework. The overall flow involves first finding out which features are correlated to the job runtime, and then use only those features for training (and testing) a regression model.

During the training phase, we first encode all the input features of all of the samples of the training data using one of the target encoding options. These encoded features are then used to compute the Pearson correlation coefficient (p) w.r.t. job runtime (represented as scontrol.RunTime). We select those features that show a Pearson coefficient of greater than 0.1. This allows us to cover a wide range of correlated features from low correlation to high correlation. The correlated features are then encoded and used as input to train a regression model to predict runtime.

In the testing phase, we simply only use the correlated features (determined during training phase) of each of the test sample for prediction of runtime. These correlated features for each sample are encoded using an encoding scheme, followed by input to the pre-trained regressor model which then predicts the runtime.

In the remaining section, we describe the various encoding options we explored in this paper. For each encoding, we provide a brief description and also the pros and cons of using that encoding.

Label/Categorical encoding assigns each string feature an integer (e.g., "alice"→0, "bob"→1), which is fast and memory-light but imposes arbitrary ordering and loses inter-category relationships; one-hot encoding maps string features to orthogonal binary vectors, avoiding artificial order but causing high dimensionality and slower training while still missing relationships; SBERTs encode both numeric and string features as a single N-dim vector: SBERT `all-MiniLM-L6-v2` (6-layer, 384-dim) yields compact, fast embeddings but lower capacity (88 MB); `all-MiniLM-L12-v1` (12-layer, 384-dim) improves semantic fidelity with modest overhead (129 MB); `all-mpnet-base-v2` (MPNet, 12-layer, 768-dim) offers state-of-the-art embedding quality at a heavier footprint (421 MB); and `xlm-r-base-en-ko-nli-ststb` (XLM-R, 12-layer, 768-dim) provides multilingual, richer embeddings with the highest memory cost (1.1 GB).

In this paper, we explore the effects of our choices of: (i) which encoding to use when encoding all the features during the process of determining the variables correlated to job runtime, and (ii) after we have determined the correlated features, which encoding to use to encode those correlated features before feeding them to a regressor model. As shown above, we have six choices of encoding; each of the six can be used during the extraction of correlated features, then each of the six can be used to encode correlated features before performing regression. Hence, we have a total of 36 different combinations that we evaluate to find the best combination.

3 Predicting Job Runtime

3.1 Experimental Methodology

Dataset. Our dataset consists of 90,000 Slurm job logs where each log contains 169 Slurm parameters. The target of our prediction is RunTime. The dataset is split into 80% training and 20% test set. To

Table 1: Features selected for each encoding method, where Pearson $r > 0.1$ w.r.t. `RunTime`.

Encoding Method	Correlated Features
Label/Categorical Encoding	NtasksPerN:B:S:C, NumTasks, SubmitTime, UserId, CPUs/Task, NumCPUs, NumNodes, MinCPUsNode, BatchFlag, TimeLimit, MailUser, TRES, GroupId
One-hot Encoding	NumTasks, CPUs/Task, NumCPUs, NumNodes, Partition, MinCPUsNode, BatchFlag
SBERT `all-MiniLM-L6-v2`	MailType, JobName, NumTasks, CPUs/Task, NumCPUs, WorkDir, NumNodes, MinCPUsNode, BatchFlag, Licenses
SBERT `all-MiniLM-L12-v1`	MailType, NumTasks, CPUs/Task, NumCPUs, NumNodes, Partition, Command, MinCPUsNode, BatchFlag, TimeLimit, Licenses, MailUser, TRES
SBERT `xlm-r-base-en-ko-nli-ststb`	NtasksPerN:B:S:C, JobName, Nodes, NumTasks, CPUs/Task, NumCPUs, WorkDir, NumNodes, Partition, Command, MinCPUsNode, BatchFlag, Licenses, MailUser
SBERT `all-mpnet-base-v2`	NumTasks, BatchScript, CPUs/Task, NumCPUs, WorkDir, NumNodes, Command, MinCPUsNode, BatchFlag, TimeLimit, TRES

evaluate the generalization of our models, we perform a 5-fold cross validation, where we randomly select the 80-20 split for 5 different folds. For each fold, we compute the prediction accuracy in terms of r^2 score and report the average r^2 across the 5 folds.

Data preparation. Before we perform the correlation analysis and use the correlated features for regression, we perform the following data pre-processing steps: (i) remove those features that contain empty or NaN entries in at least one of the 90000 samples. This typically removes features that are not important; (ii) remove those features from the log files which are not available to the job scheduler at job scheduling time. For example, ExitCode is present in the log files but will not be available to the job scheduler as it is set after a job is completed; and (iii) before we train a regression model using the encoded features, we normalize these features using the standard scaler technique.

Encodings and regression models. We compare the six different encodings as presented in Section 2. The encoded features are used as input by four different regression models: (i) Random Forest (RF); (ii) XG Boost (XG); (iii) AdaBoost (AD); and (iv) a multi-layer perceptron neural network (NN).

3.2 Results

Table 1 presents the job features that are correlated with the target variable RunTime (Table 3 in Appendix describes these features). As evident, the use of different encodings to determine the correlation (i.e., the ones that show Pearson_r Coefficient > 0.1) lead to the selection of varied features. It's important to note that when determining the correlation coefficients, except for the Label/Categorical encoding, we needed to apply Principal Component Analysis (PCA) to reduce the dimensionality of the encoded variables to one (as both One-hot and SBERT encodings lead to multi-dimensional arrays for each encoded feature). We then calculate the correlation coefficient w.r.t. runtime for One-hot/SBERT encodings, which leads to some loss of information due to PCA. On the other hand, Label encoding does not require PCA as each feature is encoded as a distinct integer. As shown in the table, most of these sets of correlated features include the usual suspects such as `NumTasks, SubmitTime, UserID, CPUs/Task,` and `NumCPUs.` Some also contain the lesser known features such as `MailUser.`

Table 2 shows the average r^2 results (over 5 folds) for different cases depending on which encoding is used to determine the features correlated to runtime, and then varying the different encodings to encode those correlated features. r^2 is reported for all four regression models that take in the encoded correlated features and output the predicted runtime.

Table 2 shows that when we use Label/Categorical encoding to encode all the features in the dataset in order to determine the correlated features, we find that using SBERTs all-mpnet-base-v2 and all-MiniLM-L6-v2 to then encode those extracted correlated features leads to the best r^2 of 0.88 with

Table 2: Average r^2 (5-fold) for combinations of encoding and regression methods.

(a) Feature selection encoding method: Label/Cat., One-Hot, SBERT MiniLM-L6

Encoding correlated features using:	Label/Cat. (best = 0.88)				One-Hot (best = 0.58)				SBERT L6 (best = 0.82)			
	RF	XG	AD	NN	RF	XG	AD	NN	RF	XG	AD	NN
Label/Cat.	0.16	0.06	0.13	0.38	**0.58**	**0.58**	0.18	0.20	0.28	−0.13	−0.01	0.00
One-hot	0.26	0.04	0.24	0.34	0.54	0.41	0.07	−0.68	0.36	−0.21	0.00	0.30
MiniLM-L6 (88 MB)	**0.88**	0.86	0.13	0.69	**0.58**	**0.58**	0.22	0.49	0.81	**0.82**	−0.06	0.69
MiniLM-L12 (129 MB)	0.87	0.85	−0.05	0.67	**0.58**	**0.58**	0.26	0.52	0.81	0.81	0.02	0.72
MPNet (421 MB)	**0.88**	0.87	0.23	0.71	**0.58**	0.57	0.22	0.55	0.81	**0.82**	0.04	0.73
XLM-R (1.1 GB)	0.86	0.84	0.02	0.71	**0.58**	0.57	0.27	0.52	0.78	0.78	0.18	0.69

(b) Feature selection encoding method: SBERT MiniLM-L12, SBERT XLM-R, SBERT MPNet

Encoding correlated features using:	SBERT L12 (best = 0.85)				SBERT XLM-R (best = 0.82)				SBERT MPNet (best = 0.85)			
	RF	XG	AD	NN	RF	XG	AD	NN	RF	XG	AD	NN
Label/Cat.	0.19	0.09	0.12	0.43	−0.20	0.13	0.06	0.14	0.08	0.07	0.18	0.37
One-hot	0.33	0.00	0.27	0.18	0.36	0.24	−0.23	0.06	0.29	0.07	0.26	0.28
MiniLM-L6 (88 MB)	**0.85**	**0.85**	−0.01	0.74	**0.82**	0.80	−0.01	0.71	0.82	0.82	0.25	0.70
MiniLM-L12 (129 MB)	0.84	0.84	0.26	0.75	0.81	0.79	−0.09	0.69	0.82	0.82	0.26	0.70
MPNet (421 MB)	0.84	**0.85**	0.37	0.76	0.78	0.77	−0.04	0.70	0.84	**0.85**	0.11	0.73
XLM-R (1.1 GB)	0.78	0.78	0.39	0.73	0.73	0.72	0.00	0.64	0.80	0.80	0.37	0.69

RF model. It is important to note that while these models provide similar accuracy, all-MiniLM-L6-v2 takes significantly lower memory (88 MB) than MPNet (421 MB) which has a direct impact on the time taken to encode the correlated variables during job scheduling. On the other hand, the use of Label/Categorical encoding on the correlated features (which is commonly used in literature), we observe that NN model shows a r^2 of 0.3821 which is $2.3\times$ lower than the best encoding all-MiniLM-L6-v2 (0.88). It is notable that while the SBERT model XLM-R is the most complex model we explored (memory size of 1.1 GB), it did not perform well compared to the other smaller SBERTs for our task of job runtime prediction. Section 4 in Appendix provides more detailed analysis of the correlated features obtained through Label encoding.

The remaining results in Table 2 show the impact of using the rest of the encodings for determining the correlated features. The r^2 values are lower than the best correlated feature configuration obtained with Label encoding. In all these cases also, we see the SBERTs are performing better than the Label/Categorical or One-hot encodings in terms of encoding the correlated features. Label/Categorical encoding is the best during extraction of correlated features as it does not cause any loss of information (no PCA applied) during the computation of correlation coefficients, which is the case for these remaining results. Note that no such dimensionality reduction is required when encoding the correlated features that are input to the regression models, hence showing the BERTs performing better than Label or One-hot for all of these cases. Finally, we find RF and XG models are in general performing better than the other regression models for our dataset.

4 Conclusion

This work demonstrates that Transformer-based semantic encoding of job features can significantly improve the accuracy of machine learning models for runtime prediction compared to traditional encoding schemes. Among the models evaluated, all-mpnet-base-v2 and all-MiniLM-L6-v2 delivered the highest predictive performance, but the latter model leads to reduced computational overhead. We found that adopting NLP-inspired encoding techniques can offer tangible benefits for intelligent scheduling in HPC environments. This work opens new directions for optimizing job management strategies through advanced representation learning.

References

Francesco Antici, Andrea Borghesi, and Zeynep Kiziltan. Online job failure prediction in an HPC system. In *European Conference on Parallel Processing*, pages 167–179. Springer, 2023a.

Francesco Antici, Keiji Yamamoto, Jens Domke, and Zeynep Kiziltan. Augmenting ML-based predictive modelling with NLP to forecast a job's power consumption. In *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, pages 1820–1830, 2023b.

Fengxian Chen. Job runtime prediction of HPC cluster based on PC-Transformer. *The Journal of Supercomputing*, 79(17):20208–20234, 2023.

Kenneth Lamar, Alexander Goponenko, Omar Aaziz, Benjamin A Allan, James M Brandt, and Damian Dechev. Evaluating HPC job run time predictions using application input parameters. In *Proceedings of the 17th ACM International Conference on Distributed and Event-based Systems*, pages 127–138, 2023.

Kevin Menear, Ambarish Nag, Jordan Perr-Sauer, Monte Lunacek, Kristi Potter, and Dmitry Duplyakin. Mastering HPC runtime prediction: From observing patterns to a methodological approach. In *Practice and Experience in Advanced Research Computing 2023: Computing for the Common Good*, pages 75–85. 2023.

Kevin Menear, Kadidia Konate, Kristi Potter, and Dmitry Duplyakin. Tandem predictions for HPC jobs. In *Practice and Experience in Advanced Research Computing 2024: Human Powered Computing*, pages 1–9. 2024.

Nils Reimers. all-minilm-l12-v1. `https://huggingface.co/sentence-transformers/all-MiniLM-L12-v1`, 2020a. Accessed: 2025-07-17.

Nils Reimers. all-mpnet-base-v2. `https://huggingface.co/sentence-transformers/all-mpnet-base-v2`, 2020b. Accessed: 2025-07-17.

Nils Reimers. all-minilm-l6-v2. `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`, 2021a. Accessed: 2025-07-17.

Nils Reimers. xlm-r-base-en-ko-nli-ststb. `https://huggingface.co/sentence-transformers/xlm-r-base-en-ko-nli-ststb`, 2021b. Accessed: 2025-07-17.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 3982–3992. Association for Computational Linguistics, 2019. URL `https://aclanthology.org/D19-1410`.

Srividya Srinivasan, Rajkumar Kettimuthu, Vijay Subramani, and Ponnuswamy Sadayappan. Characterization of backfilling strategies for parallel job scheduling. In *Proceedings. International Conference on Parallel Processing Workshop*, pages 514–519. IEEE, 2002.

Chiara Vercellino, Alberto Scionti, Giuseppe Varavallo, Paolo Viviani, Giacomo Vitali, and Olivier Terzo. A machine learning approach for an HPC use case: The jobs queuing time prediction. *Future Generation Computer Systems*, 143:215–230, 2023.

Wenxiang Yang, Xiangke Liao, Dezun Dong, and Jie Yu. Exploring job running path to predict runtime on multiple production supercomputers. *Journal of Parallel and Distributed Computing*, 175:109–120, 2023.

Appendix

As supplementary material to support the main text, this appendix acknowledges support for this work, and gives descriptions of the Slurm features selected by Label encoding based correlation analysis so readers can interpret effects of various features. Insight into the Pearson correlation calculation is provided and then related work is described in more detail.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Feature descriptions

Table 3 describes the various correlated features of Table 1 in more detail.

Table 3: Description of correlated Slurm features of Table 1.

Feature	Description
NtasksPerN:B:S:C	Tasks placement breakdown per Node:Board:Socket:Core (from `-ntasks-per-*` directives).
NumTasks	Total number of tasks requested.
SubmitTime	Timestamp when the job was submitted.
UserId	UID or username of the job owner.
CPUs/Task	CPU cores allocated to each task (`-cpus-per-task`).
NumCPUs	Total CPU cores = NumTasks $\times$ CPUs/Task.
NumNodes	Nodes allocated to the job.
MinCPUsNode	Minimum CPU cores per node.
BatchFlag	Indicates batch (non-interactive) mode.
TimeLimit	Wall clock time limit for the job, either user specified or set by Slurm if not specified (HH:MIN:SECS).
MailUser	Email address for status notifications.
TRES	Consolidated trackable resources (cpu, mem, gres, etc.).
GroupId	Accounting group or project ID.
Partition	Queue (partition) where the job was submitted.
MailType	Events triggering email (BEGIN, END, FAIL).
JobName	User defined name of the job.
WorkDir	Working directory for the job.
Licenses	Software licenses requested.
Command	Command or script executed by Slurm.
Nodes	Alias for NumNodes.
BatchScript	Path to the submission script.

Correlated features obtained using integer encoding and insights

Figure 2 provides a more detailed look at the correlated features obtained through Label/Categorical encoding.

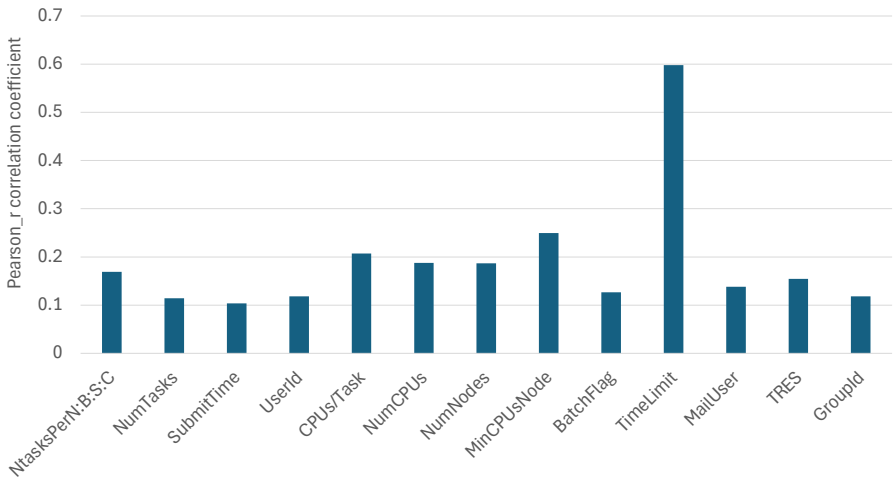

Figure 2: Pearson_r correlation coefficient for features selected using the Label/Categorical encoding method in Table 1.

Below, we provide the key insights we gained from this analysis about our dataset.

1. **TimeLimit.** Users often choose `TimeLimit` close to their expected wall time or with a consistent bias. If they do not specify this feature, Slurm sets the limits based on resource limits. Therefore, `TimeLimit` shows the strongest linear relationship with actual `RunTime`.
2. **Proxies for job size.** Features like `MinCPUsNode`, `NumCPUs`, `NumNodes`, `CPUs/Task`, and `NumTasks` scale with requested resources and typically with problem size, yielding positive correlations with `RunTime`. These variables are also highly collinear.
3. **Packing and placement directives.** `NtasksPerN:B:S:C` reflects how tasks are packed across nodes, boards, sockets, and cores. Jobs needing finer placement often run longer, but the effect is weaker than for `TimeLimit`.
4. **Administrative and bookkeeping signals.** Fields like `TRES`, `BatchFlag`, `MailUser`, `UserId/GroupId`, and `SubmitTime` capture policy or user behavior patterns, producing modest correlations that reflect usage trends rather than causation.

Related work

There has been valuable work toward predicting job runtimes using machine learning. These papers generally use traditional encodings and do not explore the effects of different encodings on prediction accuracy. Consequently, selecting the optimal encoding method is not covered.

A recent paper explored the use of job features, such as number of CPUs and number of nodes requested, to predict job queuing time using both supervised and unsupervised ML techniques Vercellino et al. [2023]. They formulate the problem as both a classification task, where targets are binned, and a direct regression task. Lasso regression is used to select relevant features, which are then used by ML models such as K nearest neighbors and neural networks. However, details are not provided about how features were encoded before training. Similarly, another paper used job running path as an important feature in addition to typical features such as `reqCPUs`, `UserID`, and `JobName` for runtime prediction Yang et al. [2023]. They used hash encoding for categorical features but did not perform correlation analysis. Studies have also combined features from queue state and resource utilization for runtime prediction Menear et al. [2024]. This work explored one hot and integer based label encoding. Additionally, techniques that use application features such as LAMMPS or FFT parameters with standard job parameters for runtime prediction have been explored Lamar et al. [2023]. The non numeric or categorical features are encoded using one hot encoding.

While the above work used more traditional models such as random forest and multi layer perceptron, others have explored Transformer models. For example, Chen [2023] pre processes job data in a time ordered sequence to capture dependencies between jobs. They replace the normal decoder of the Transformer with a linear layer to predict runtime.

Two papers use SBERT to encode features from job submission scripts. The encoded features are then fed to other ML algorithms such as random forest, decision trees, and k means clustering. However, they do not predict job runtime: one paper predicts job failure Antici et al. [2023a] and the other predicts power consumption Antici et al. [2023b].